VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2018 Vladimír Gromotovič

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Vladimír Gromotovič**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Pavel Dohnálek**

Konzultant bakalářské práce: MSc. Artur Banczyk

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 25. dubna 2018 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

Tieto Czech s.r.o.
28. října 3346/91
702 00 Ostrava - Moravská Ostrava
IČO 64608051 DIČ CZ64608051

V Ostravě 25. dubna 2018

.....

Rád bych na tomto místě poděkoval Ing. Pavlu Dohnálkovi a mému týmu za pomoc a ochotu při mé odborné praxi, bez jejichž pomoci by tato práce nevznikla.

Abstrakt

Tato práce se zabývá průběhem odborné praxe ve firmě Tieto Czech s.r.o. na pozici Junior java developer. Věnuje se řešení běžných úkonů developera, které byly studentovi zadány v průběhu odborné praxe. Na konci práce je zhodnocení nabytých zkušeností s prací v týmu nad komerčním projektem v nadnárodní společnosti.

Klíčová slova: praxe, webová aplikace, Tieto, Java, AngularJS, Hibernate

Abstract

This thesis deals with the course of professional practice in Tieto Czech s.r.o. as Junior java developer. It deals with the solution of the routine tasks of the developer which were assigned to student during professional practice. At the end of the work is an evaluation of the experience gained in a team over a commercial project in a multinational company.

Key Words: practice, web application, Tieto, Java, AngularJS, Hibernate

Obsah

Seznam použitých zkratek a symbolů

UML	–	Unified Modeling Language
HTML	–	Hyper Text Markup Language
CSS	–	Hyper Text Markup Language
SVN	–	Subversion
SQL	–	Structured Query Language
R&D	–	Research and Development
CRM	–	Customer relationship management
HTTPS	–	HTTP Secure - Hypertext Transfer Protocol
ORM	–	Object-relational mapping
JVM	–	Java virtual machine
CRUD	–	Create, read, update, delete
IDE	–	Integrated development environment
XML	–	Extensible Markup Language
JSON	–	JavaScript Object Notation
DTO	–	Data transfer object
DAO	–	Data access object
Frontend	–	Část aplikace běžící na klientovi
Backend	–	Část aplikace běžící na serveru

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Při výběru tématu bakalářské práce hrála roli možnost ji a absolvovat formou praxe ve firmě. Vybral jsem si Tieto Czech s.r.o., ve které jsem chtěl nejen získat praktické zkušenosti jako vývojář, ale také zvýšit svou hodnotu na trhu práce a zároveň posoudit jestli mi tato práce vyhovuje a naplňuje mě. Zjistit jaké technologie a postupy se využívají v praxi a poznat vývojáře z různých zemí, kteří mají několikaletou praxi s vývojem aplikací a projektů a prošli různýma firemníma prostředí.

Na začátku uvedu informace o firmě a týmu, do kterého jsem nastoupil. Dále popíšu technologie, které jsem v rámci praxe využíval na denní bázi. Dále se budu věnovat úkolům, které mi byli svěřeny a jejím řešením. Na závěr zhodnotím, jak jsem byl připraven na prostředí reálného komerčního projektu a jaké znalosti, převážně z vysoké školy, jsem při své praxi využil a jaké znalosti mi naopak chyběly.

2 Představení firmy

2.1 O firmě

Společnost Tieto byla založena ve Finském městu Espoo v roce 1968, sloužila především jako počítačové centrum pro vlastníky. Informační systémy byly vyvíjeny a udržovány hlavně pro Finskou Union Bank a pro několik lesnických společností. Finské Tieto a švédská společnost Enator se spojili 7. července 1999. Od 26. března 2009 nese společnost název Tieto Corporation. Tieto tvoří většinu produktů pro skandinávský trh.

Na český trh Tieto vstoupilo v roce 2001 a v roce 2004 otevřelo softwarové centrum v Ostravě. V roce 2012 dostavěli vlastní kanceláře nazvané Ostrava Tieto Towers. Tieto Czech v roce 2017 zaměstnává přes 2300 kmenových zaměstnanců, což z nich dělá jednoho z největších zaměstnavatelů v oblasti IT služeb v České republice a největšího v rámci Moravskoslezkého kraje.[1, 2]

2.2 Media Solutions/R&D tým

Tým, ve kterém jsem absolvoval individuální odbornou praxi, se zabývá jak internetovou tak i tištěnou reklamou a to ve formě novin. Momentálně se vyvíjí jediný produkt a to Cross-advertisment, zkráceně CrossAd. V tomto týmu je přes 30 lidí, ve složení - projektový manažer, obchodní ředitel, konzultanti, technici, databázisti, vývojáři a testeři. Jádro týmu se nachází ve Švedsku. V Ostravě je větší čast týmu a je složený převážně z vývojářů a testerů.

Tým využívá aplikace firmy Atlassian a to Jira Software a Confluence. V rámci Tieta se používá pro komunikaci aplikace Microsoftu a to Lync/Skype for business a Outlook a služby Googlu, na které se pomalu přechází ze služeb od Microsoftu. Jira Software je komplexní webové řešení na hlášení chyb, změn, přidávání nových funkcích apod. Confluence je webová aplikace obsahující dokumentaci, informace o týmu a projektech. Příma komunikace je sice nejspolehlivější a nejrychlejší volba, ale pokud člen týmu je ve Finsku, Švedsku nebo na home office tak je potřeba komunikovat pomocí softwaru a k tomu se používá zmíněný Lync/Skype for business pro komunikaci v reálném čase a Outlook převážně pro informativní emaily.

2.3 Cross-advertising platforma

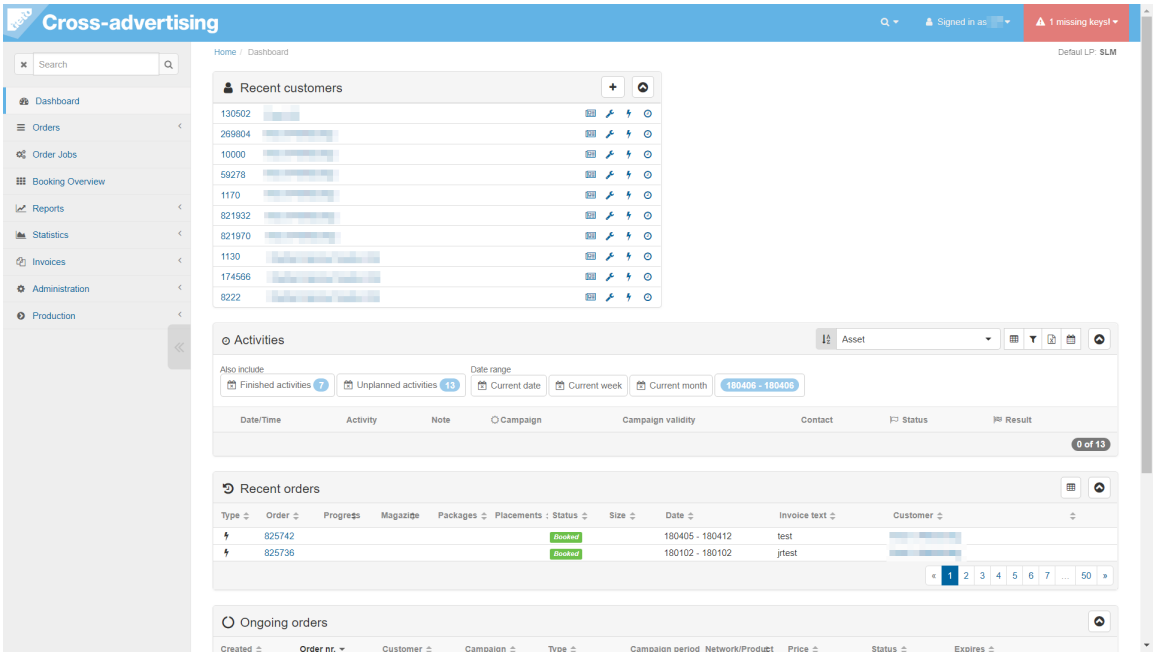
Platforma slouží jako řešení pro komplexní správu reklamy, která pokrývá celý proces od prodeje, rezervace a fakturace. Hlavní funkce platformy, CRM - jedná se o zákaznicky orientované řízení, které řídí celý cyklus komunikace se zákazníkem, produkce reklamy, nástroje na plánování reklamy, řízení financí a fakturací, administrace produktů - umístění, ceníky, slevy, balíčky a sítě. Platforma obsahuje nástroje pro integraci se systémy třetích stran jako jsou redakční, finanční a ad-servery [3].

Software je vytvořen ve 3 verzích, desktopová(standard edition), mobilní(mobile edition) a webová aplikace(web edition). Mobilní aplickace byla vytvořena na technologii jQuery a už se

dále nevyvíjí a ani nepodporuje, byla zcela nahrazena aplikací webovou. Desktopová aplikace byla vytvořena na platformě Omnis Studio a aplikace je pořád podporováná, ale už se dál nevyvíjí a do budoucna by měla být zcela nahrazena webovou aplikací. Webová aplikace je nejaktuálnější iterací této softwarové rodiny, která je podporovaná a vyvíjená. Webovou aplikaci lze používat na platformách Windows, MacOS a mobilních platformách Android, iOS.

2.3.1 Web edition

Jedná se o webovou aplikaci. Lze ji použít na jakékoliv platformě s webovým prohlížečem, včetně platformy mobilní. Ale oficiálně podporované platformy jsou Windows, MacOS, iOS, Android a prohlížeče Chrome, Firefox, Opera, Safari. Webová aplikace je stále vyvíjená a podporovaná ve třech hlavních verzích 7.4.5.x, 7.5.0.x a 7.5.x. Momentálně stále plně nenahrazuje standard edici. Používá se jenom na ulehčení a urychlení nejpoužívanějších procesů ze standard edice. Ve verzi 8.0 by web edice měla plně nahradit standard edici. Tato verze je plánovaná na konec roku 2018.



Obrázek 1: Úvodní stránka webové aplikace ve verzi 7.5.x

2.3.2 Architektura

Frontend je postaven na technologiích HTML5, AngularJS a CSS knihovně Bootstrap. Backend běží na Javě a jádro je tvořeno ve frameworku Spring a data jsou uchovávaná v SQL databázi OracleDB. ORM zajišťuje framework Hibernate. Komunikace mezi frontendem a backendem probíhá přes HTTPS protokol na server Tomcatu. Dále se využívá Maven pro sestavení projektu a přidání závislostí na knihovnách a frameworků třetích stran a SVN pro správu a verzování zdrojových kodů aplikace.

Zákazník potřebuje vlastní server pro backend aplikace a server pro databázi, o které se starají technici z týmu. Protože správa těchto serverů je finančně náročná jak pro zákazníky, tak i pro tým, proto je ve vývoji cloudové řešení, které bude operovat na serverech Amazonu a bude tak finančně méně náročné.

3 Použité technologie

3.1 Java

Objektový programovací jazyk, který vznikl ve společnosti Sun Microsystems v roce 1995. Přes 15 miliard zařízení obsahuje Javu. Jedná se o stolní počítače, mikropočítače a telefony. Převážně se používá v korporátním prostředí, než v prostředí malých projektů. To se ale pravděpodobně změní s příchodem IoT - Internet of Things. Java je interpretovaný jazyk, kód se překladá do mezikódu tzv. bytecode a ne přímo do strojového kódu. Tento bytecode je poté přeložen virtuálním strojem Javy tzv. JVM, tento virtuální stroj mají všechna zařízení obsahující Javu. Díky tomuto překladu do mezikódu není potřeba měnit kód aplikace, ale jen JVM pro dané zařízeni. Proto se jedná o nezávislý jazyk na platformě. Narozdíl od některých jazyků se o pamět stará Garbage Collector a ne programátor. To znamená, že nemůže dojít k úniku paměti a to je jeden z důvodů, proč se pro projekt zvolila právě Java. Podobnou funkcionalitu obsahuje například i .NET, ale ten v té době běžel pouze na Windows platformách a bylo by nutné platit licence za Windows server. Kdežto Java běží na platformách jako je Linux, který je zdarma i pro komerční použití. [4, 5]

3.2 AngularJS

Jedná se o javascriptový open source framework vyvíjen převážně společností Google postavený na jQuery lite knihovně. Dnes už AngularJS není ve vývoji a firma Google se soustředí na vývoj Angularu 5, který je značně odlišný od AngularuJS. Pro projekt byl zvolen, protože byl ve své době hodně rozšířený a byl vyvíjen velkou společností. Zpětně tato volba nebyla z pohledu týmu nejlepší, jelikož javascript je interpretovaný jazyk a dynamicky typovaný a je značně odlišný od kompilovaného a staticky typovaného jazyka Java. Dnes by se volil open sourcový framework Typescript vyvíjený Microsoftem, který přidává statické typování a objektově orientované prvky jiných jazyků. Samotný kód se poté kompiluje do javascriptu. [6, 7]

3.3 Hibernate

Jeden z největších ORM frameworků pro Javu. Zajišťuje mapování mezi objekty Javy a tabulkami v databázi, také zajišťuje CRUD - základní operace nad tabulkami. V průběhu vývoje naší aplikace se vyvíjel také tento framework a byl nasazen Hibernate 3 za Hibernate 2. V Hibernate 3 se například změnilo mapování tabulky na objekt. Dříve toto mapování bylo prováděno přes XML soubor. Nyní se toto mapování nahradilo za anotace. Jsou přehlednější, ale je nutné si uvědomit, že se potom s tímto mapováním pracuje trochu odlišně a je potřeba si na to dát pozor.

3.4 Ostatni

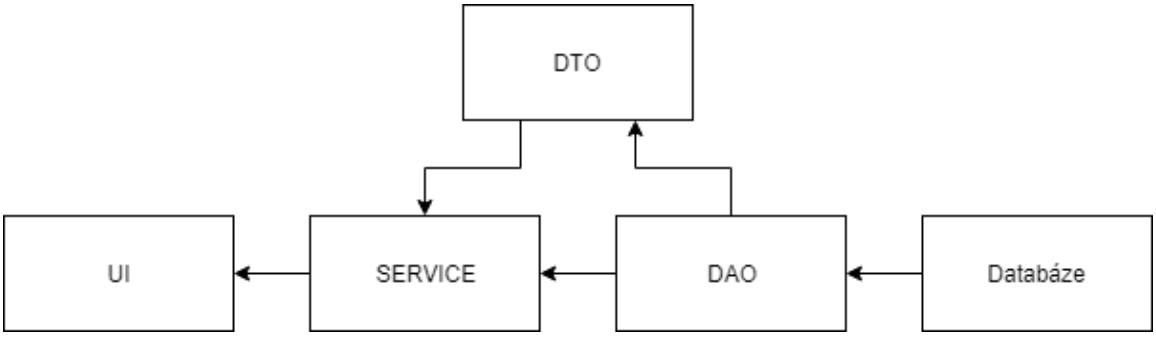
Další z technologií je SVN, která slouží pro verzování aplikace a obsahuje plno dalších funkcí jako je merge, což je vytvoření kopie změny a přidání do jiné verze/větve - branch. Tato technologie byla zvolena pro její jednoduchost a hlavně kvůli praktickým zkušenostem s touto technologií, kterou tým měl. V projektu se používá plno dalších technologií, frameworků a knihoven. Jeden z menších je například JasperReports, který se v projektu používá na generování pdf a excel souborů. Dále bych chtěl zmínit technologie, které ulehčují samotné programování, jako je IDE. Já jsem používal IntelliJ od JetBrains, ale v týmu se používa take Eclipse a Netbeans, především jde o preferenci každého vývojáře. Pro HTML a javascript jsem používal Visual Studio Code. IntelliJ přináší funkci našeptávání pro programování v Javě, přehlednost a indexování souborů v projektu, které mi umožňuje rychlé vyhledávání v projektu. Nejdůležitější funkcí je, ale debugování aplikace, které je klíčové pro hledání chyb. Dále IntelliJ je propojená s SVN a sleduje změny, které jsem udělal ve zdrojových kódech. Protože je SVN propojeno s IntelliJ, nepotřebuji žádný desktopový klient SVN, jako je například TortoiseSVN.

4 Úkoly zadané během praxe

Jak už bylo dříve napsáno nastoupil jsem do firmy jako Junior Java developer, kde jsem pracoval na frontendu(javascript, HTML) a backendu(Java a její frameworky, SQL dotazy). Žádný z úkolů nepotřeboval k dokončení databázové změny. Před začátkem jakéhokoliv úkolu bylo potřeba si nastavit vývojové prostředí, nainstalovat si IDE, nastavit lokální Tomcat server a nainstalovat lokální databázi. Jak jsem sám zjistil tak testovat funkci s daty z databáze, která je umístěna ve Švédsku, trvá třikrát déle než s lokální databází, kde prakticky není žádná prodleva. V této sekci popisuji několik nejzajímavější úkolu, které jsem během odborné praxe dostal. Převážně mi byli svěřovány menší úkoly, na kterých jsem se naučil orientovat v projektu.

4.1 Implementace chybějících funkcí

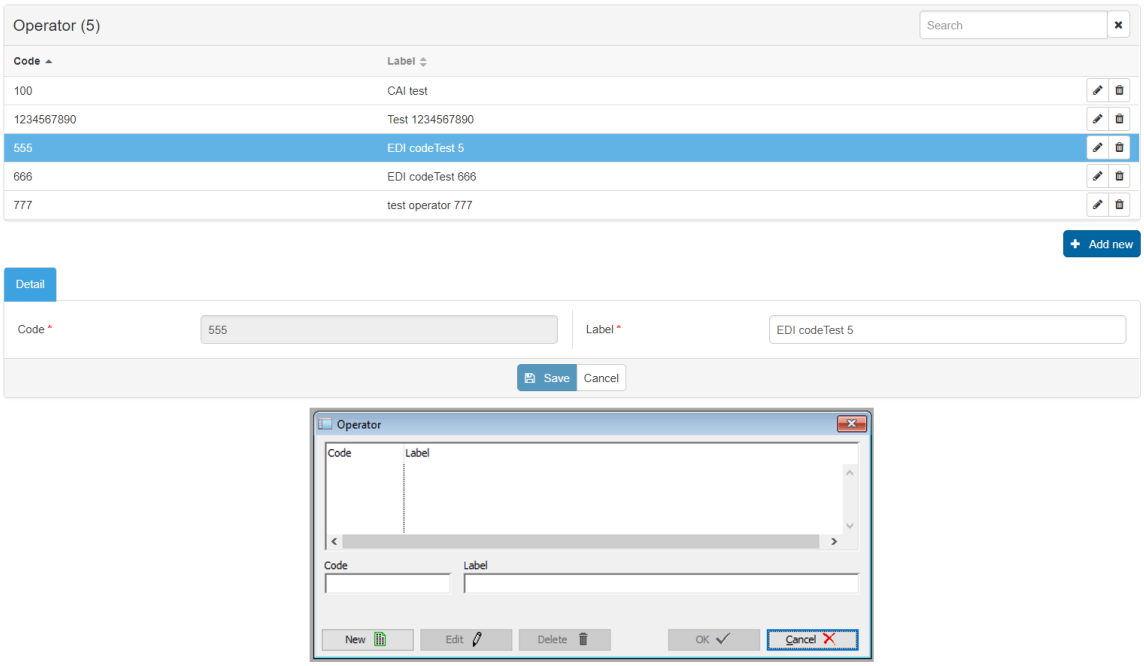
Můj první úkol byl implementace funkcí ze standard edice do web edice. Jednalo se o administraci Operator, Production status a Production codes, detailní popis jak tyto funkce fungují byl v dokumentaci standard edice přiložený k úloze. Začal jsem s funkcí Operator, jednalo se o číselník, který se přidával k zákazníkovi. Našel jsem si podobný číselník a podíval jsem se jak vypadá a podle toho jsem si vytvořil HTML šablonu. Web edice a standard edice sdílí stejnou strukturu databáze. Jelikož tato funkce je implementace ze standard edice, tak tabulky v databázi jsem měl už připravené. Také byla vygenerovaná ORM třída od Hibernatu k tabulce. Pokračoval jsem vytvořením tří vrstev DTO, DAO a service v Javě. Výhoda této architektury je, že DAO může přistupovat k různým databázím nebo datům, např. XML, MySQL, OracleDB a další, ale service vrstva je od tohoto přístupu k datům distancovaná. DTO vrstva je objekt, který uchovává data



Obrázek 2: DTO, DAO a service vrstva

mezi procesy, DAO zprostředkovává komunikaci se zdrojem dat. V tomto případě s SQL datábazí, jelikož dědí z Hibernatí třidy, tak obsahuje základní metody nad daným zdrojem dat, findByID, save, delete, getList. Pro tento úkol mi stačily tyto základní metody. Poslední vrstva je service vrstva. Zprostředkovává metody, které jsou potřeba pro daný číselník - viz. obrázek č.3. Jedna z metod je smazat operátor, tady bylo potřeba ošetřit případ, že je operátor využíván. Musel jsem si od testera zjistit, kde se operátor přidává k zákazníkovi a přidat testovací operátor. Poté pomocí chrome dev tools jsem zjistil HTTPS request a ten mě zavedl do Angulařího controlleru

a na service vrstvu, odkud jsem zjistil, v jaké tabulce jsou data uložena. Pak jsem mohl v metodě zkontrolovat, jestli daný operátor je nevyužitý a popř. ho smazat. Nakonec bylo potřeba vytvořit Angulaří controller, který se bude dotazovat na tyto metody ze service vrstvy na frontendu. Posledním krokem bylo otestovat, jestli všechno funguje jak má a předat to testerovi. Ten to otestuje podrobněji. Tímto byla dokončena implementace administrace Operátoru.



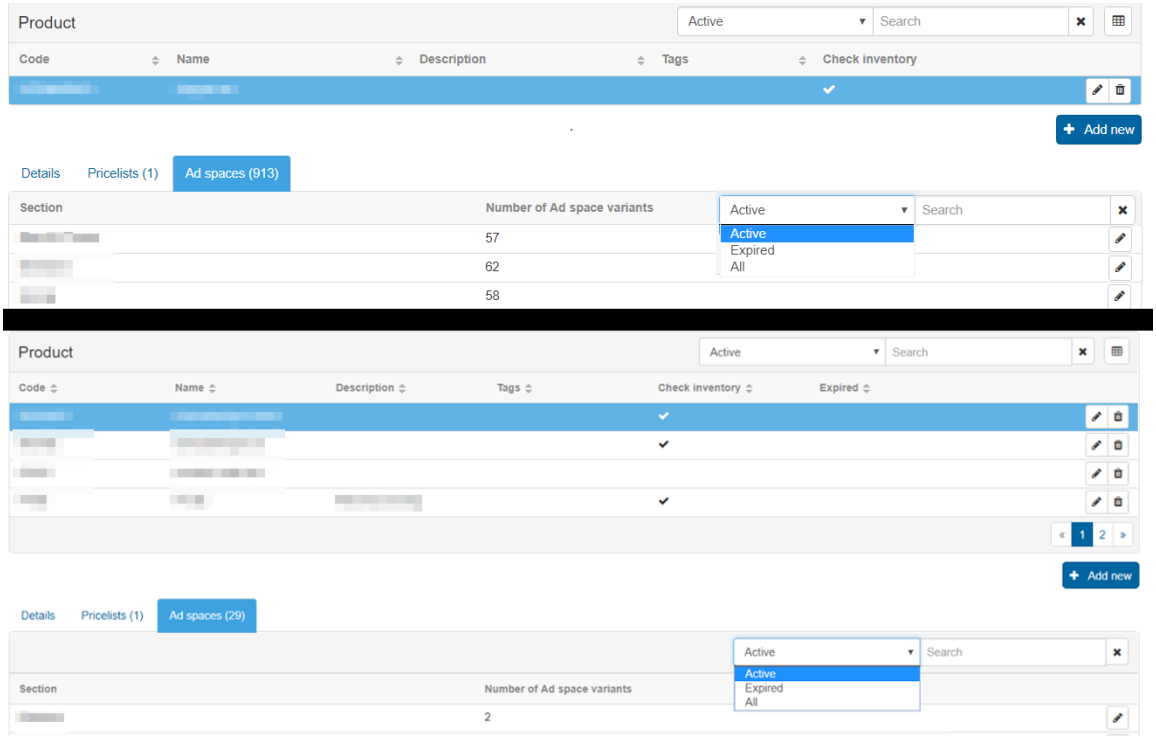
Obrázek 3: Administrace operatorů, nahoře web edice dole standard edice

Implementace funkcí Production statusu a codes, byly velmi podobné operátoru. Takže jsem tyto funkce rychle naprogramoval podle stejného postupu jako operátor, aby jsem mohl pokračovat s jinými úkoly. Odeslal jsem to testerovi na testování. Úkol se mi vrátil s několika chybami. Uvědomil jsem si, že špatné přečtení zadání úkolu nebo dokumentace vede k zpomalení vývojového procesu celé aplikace. Opravil jsem všechny vyjmenované chyby, nahrál na SVN a odeslal zpět na test.

4.2 Filtrování

Dalším z úkolů byla filtrace reklamního prostoru. Jednalo se o sekci kde inzerenti nabízeli místa pro reklamu. Tyto reklamní místa obsahovali kromě velikosti plochy reklamního prostoru, médium pro které je reklamní prostor určen, také platnost do kdy je dané místo na prodej. Tímto vznikala nepřehlednost, protože zákazníci si museli kontrolovat jestli náhodou není reklamě ukončena platnost. Byla proto potřeba přidat filtrování na aktivní, prošlé a všechny reklamní prostory. Také bylo potřeba přidat vyhledávání podle média, ve kterém je reklama umístěna. Od designera jsem dostal návrh jak by tato funkce měla vypadat.

Měl jsem dvě možná řešení. První bylo, že pošlu všechny reklamní prostory na klienta a zde budu filtrovat již prošlé inzerce. Druhá byla, že filtrování bude probíhat na serveru, tzn. klient při změně filtru bude posílat požadavek na server a ten mu vrátí výsledek. Konzultoval jsem s obchodním analytikem, jak tuto funkci zákazníci využívají a dostalo se mi odpovědi, že převážně hledají aktivní inzerce. Proto jsem se rozhodl pro druhou variantu. Prošlých inzercí může být v databázi desítky a zbytečně by zpomalovali přenos dat pro zákazníka, který hledá pouze aktivní inzerce. Takže jsem musel udělat filtr na frontendu pro slovní vyhledávání názvu média a filtr na backendu pro vrácení aktivních, prošlých a nebo všech inzercí. Na frontendu filtr byl jednoduchý, protože jsem už podobný vytvářel při úkolu implementace funkcí ze standard edice do web edice. Filtr na backendu se skládal z jednoduchého switche, ve kterých se vytvářel Criterion - jedná se o třídu Hibernatu, která tvoří pomocí metod SQL dotaz. Je to přehlednější než samotný SQL dotaz. Filtrace byla hotova, nahrál jsem upravené soubory na SVN a poslal úkol projekt manažerovy, který ho poslal volnému testerovi. Funkcionalita prošla testem bez problémů.



Obrázek 4: Nahoře - návrh, Dole - implementace

4.3 Prohlížeč přidává .txt příponu

Jedná se o soubor s příponou .sef, který se používá na exportování objednávky. Tento soubor se poté využívá v systémech třetích stran. Při stažení tohoto souboru prohlížeč přidá za příponu .sef ještě příponu .txt.

Pokusil jsem se replikovat tento problém, ale bez úspěchu, tak jsem úkol vrátil zpátky testerovi. Tester mi odepsal, že se jedná jenom o prohlížeč Safari. Hledal jsem na internetu lidi se stejným problémem. Jednalo se pravděpodobně o chybu prohlížeče Safari, ale existovaly cesty jak to spravit. Vyzkoušel jsem přidat do HTML tagu s odkazem na stahovaný soubor jeho celý název včetně přípony. Zde nastal problém, neměl jsem jak otestovat jestli daná úprava opravila chybu. Protože Safari nepodporuje Windows platformu. V týmu je jeden tester s MacOS systémem, s kterým jsem to musel řešit. Dále nastal další problém, testeři netestuji lokálně, ale přes Hudson. Hudson je systém, který pracuje s aktuálním projektem z SVN a automaticky ho builduje v určitém intervalu. Takže jsem pokaždé vyzkoušel nějaké řešení, nahrál na SVN a počkal dokud Hudson nezačne buildovat. Pak jsem to oznámil testerovi, který to otestoval. Jako poslední možnost jsem vyzkoušel přidat typ media ve formátu "text/sef". Poslal jsem to na test a chyba byla opravena. Stále si stojím za tím, že se jedná o chybu v prohlížeči Safari, nejenom z důvodu, že ostatní prohlížeče fungovali v pořádku.

4.4 Historie objednávky

Chtěl bych zde zmínit úkol, ve kterém se střetává zákazník, manažer a programátor. Zákazník chce přidat funkci, která momentálně chybí, manažer to schválí popíše problém a pošle to dál na programátory. V tomto případě se stalo to, že manažer nerozuměl správně jak daná funkcionalita funguje.

Manažer popsal nějakou funkci, kterou chce zákazník přidat. Tento úkol byl svěřený mě. Jednalo se, o přidání do historie změn objednávky údaj, o změně tzv. Custom fieldů. Custom field je dynamická složka objednávky pro její dodatečný popis. Jedná se třeba o textfield, combobox a nebo jiný HTML prvek. To by samo osobě nebyl problém, ale byl zde další požadavek a to přidat ke každému custom fieldu WHAT kód. Zeptal jsem se zkušenějšího programátora co a jak funguje WHAT kód a dozvěděl jsem se, že se jedná o statický popis, který se zobrazuje v historii změn objednávky. Úkol jsem vrátil s tím, že potřebuji informace a jsem schopen přiřadit WHAT kód pouze k danému typu custom fieldu a ne ke každému unikátnímu custom fieldu. Odpověděl mi hlavní vývojář, že funkcionalita custom field byla vytvořena s varováním, že nikdy nebude spravovaný programovou logikou. Kontaktoval jsme projekt manažera, který se mnou sepsal technickou specifikaci a úkol poslal zpět manažerovi. Tato funkcionalita je stále bez řešení.

4.5 Synchronizace překladových souborů

Jedním z posledních úkolů, který jsem v rámci odborné praxe dostal, bylo synchronizovat překladové soubory. Aplikace je překládaná do pěti jazyků. Angličtina je primární jazyk, který je plně přeložen. Dále je švédština, kde překlad už není stoprocentní a několik překladů chybí. Pak je finština, dánština a norština, ale tyto jazyky nejsou tolik podporované jako švédština. O překlady se starají rodilí mluvčí, kteří používají k překladu nástroj uvnitř webové aplikace. Zpátky k překladovým souborům. Jedná se o soubory ve formátu JSON (viz výpis 2), obsahují klíč(před

dvojtečkou) a hodnotu/překlad(za dvojtečkou), pro každý jazyk podporovaný aplikací je jeden tento soubor. Klíče se používají v HTML šablonách (viz výpis 1), AngularJS si při načítaní aplikace načte jeden ze souborů a poté daný klíč zamění za hodnotu, buď použitím direktivy nebo filtru. Rozdíl je jenom v tom, jak to funguje interně, pokud ten klíč v tom souboru existuje.

```
<div translate="global.title">Title</div> <!--ukázka použití direktivy-->
<div>{{"global.languages.en" | translate}}</div> <!--ukázka použití filtru-->
```

Výpis 1: Ukázka JSON struktury

A to je problém synchronizace souborů v našem projektu. Protože vývojáři pracují jenom s anglickým překladem, tak klíče jsou přidávány jenom do souboru s anglickým překladem. Jak aplikace narůstá a přidávají se nové klíče, tak u ostatních překladů tyto klíče chybí a HTML šablonách se nenahradí a uživatelé místo překladu vidí klíč, což dělalo prakticky jiný překlad než anglický a švédský nepoužitelný.

```
"global": {
    "title": "Cross-advertising",
    "yes": "Yes",
    "languages":{
        "en": "English",
        "fi": "Suomi",
        "sv": "Svenska".
        "da": "Dansk",
        "no": "Norsk"
    }
}
```

Výpis 2: Ukázka JSON struktury

Jelikož jsem byl v týmu už nějakou dobu a věděl jsem také z vlastní zkušenosti, že když jsem narazil na nějaký klíč v šabloně a chtěl jsem mu např. změnit hodnotu/překlad, tak najít daný klíč v souboru s 8000 řádky, který se různě zanořuje, je časově náročné. Navrhl jsem změnu formátu těchto souborů a to na properties formát (viz výpis 3). Navrhovaná změna byla vývojáři v týmu přijata. Dále jsem navrhl, že synchronizace těchto souborů bude probíhat při nahrání na SVN, jelikož jsem si zjistil, že je možné nahrát skript, který se spustí při nahraní určitých souborů. Tento návrh neobstál z důvodu pomalejšího nahrávání těchto souborů na SVN server. Z mého pohledu se nejednalo o razantní časovou prodlevu při nahrávání a zcela by to automatizovalo synchronizaci těchto souborů. Finální řešení bylo, že před releasem nové verze aplikace se tyto soubory stáhnout synchronizované z aplikace a nahradí se za původní nesynchronizované soubory a nahrají se na SVN.

Jak už jsem dříve zmínil, v aplikaci je nástroj na překlad, který je schopen doplňovat chybějící klíče a přidat k nim překlad pro daný jazyk, který tam vloží překladatel. Modifikace probíhá v

```
global.title = "Cross-advertising"
global.yes = "Yes"

global.languages.en = "English"
global.languages.fi = "Suomi"
global.languages.sv = "Svenska"
global.languages.da = "Dansk"
global.languages.no = "Norsk"
```

Výpis 3: Ukázka properties struktury

paměti na konci překladatel stáhne soubor a přepíše soubor původní a nahraje ho do SVN. Stačí si jenom najít daný skript a upravit ho tak aby přidal všechny klíče a k nim anglický překlad, kterému jsem přidal podtržítko na začátek a také jsem upravil nástroj aby nevyhledával podle chybějících klíčů, protože takové už tam nebyly, ale podle překladů, které začínají podtržítkem. Dále aplikace obsahuje nástroj, který ukazoval programátorovi při chybějícím překladu daný klíč u kterého překlad chyběl. Tady nastal menší problém, protože AngularJS a jeho funkce, který překlad zařizovala tak využívala variaci návrhového vzoru observer a oznamovala pozorovatelům jenom tehdy, když chyběl klíč, který se snažil nahradit, ale já jsem potřeboval oznámit tehdy, když nějaký překlad začinal podtržítkem. Z dokumentace od Angularu jsem zjistil, že existuje ještě jeden observer, který oznamuje pozorovatelům při korektním překladu a vrací klíč, který překládal a hodnotu klíče/překlad (viz výpis 4). Poslední krok bylo vytvořit utilitu pro vygenerování souboru properties. Tato utilita mi bude sloužit jenom pro konverzi z JSON formátu na properties formát a poté bude odstraněna. Využil jsem toho, že klíče s hodnotami se načítají do paměti, a proto stačilo jenom data naformátovat a vygenerovat soubory, které jsem potom stáhnul a zaměnil je za soubory v JSON formátu.

```
//observer, který zavolá pozorovatele při chybějícím klíči
$translateProvider.useMissingTranslationHandler('$translationKeyMissing');
//observer, který zavolá pozorovatele při správném dokončení překladu
$translateProvider.postProcess('$translationMissing');
```

Výpis 4: AngularJS translateProvider observer navrhový vzor

Všechno potřebné jsem měl hotovo a připraveno, ale nemohl jsem to jen tak nahrát na SVN, protože vývojáři, kteří upravovali původní soubory v JSON formátu a ještě je nenahráli na SVN, tak by zjistili, že daný soubor je smazaný a museli by si aktualizovat projekt a změny co udělali do původního souboru přepsat. Tento scénář je docela pravděpodobný, protože nové překlady se vkládají poměrně často. Odeslal jsem email všem vývojářům, že další den se přechází na nový formát překladů, tak ať před tímto dnem nahrají své změny, a překladatelům, kterým jsem vysvětlil změny v překládacím nástroji. Poté jsem vygeneroval properties soubory a vymazal utilitu, properties soubory nahrál na SVN a vymazal staré JSON soubory z SVN, poté jsem

vytvořil nový branch a přidal tyto změny do starší verze 7.5.0.x a to stejné jsem udělal pro verzi 7.4.5.x. V této verzi to ale nefungovalo jak mělo, protože využívala starší verzi Angularu. Kontaktoval jsem projekt manažera, kterému jsem vysvětlil problém, že bych to musel implementovat zvlášť pro tuto verzi Angularu. Projekt manažer řekl, že tato verze už nebude dlouho podporovaná a nemám tuto změnu implementovat. Nakonec jsem to odeslal testerovi na test.

Tester vrátil úkol zpět na mě s komentářem, že nástroj, který ukazuje chybějící klíče, ukazuje chybějící klíče, které mi vypsal. Prohledal jsem projekt, kde se tyto klíče nachází a proč chybí v překladech. Zjistil jsem, že tyto chybějící klíče, které nástroj ukazuje, jsou klíče vytvořené dynamicky v javascriptu a v době prvního překladu je známá jenom část klíče a AngularJS pracuje jako observer a čeká na změnu v $scope(Jedná se o část Angularu, která provazuje HTML šablonu s Angulařím controllerem). Pozval jsem si na konzultaci frontend vývojáře, který pracuje s Angularem několik let a došli jsme k závěru, že nemá cenu momentálně řešit, aby nástroj, který ukazuje tyto chybějící překlady je přestal ukazovat, protože řešení by bylo příliš komplexní na to, že tento údaj se zobrazuje jenom ve vývojářské verzi. Zkonzultoval jsem to ještě s projekt manažerem, ten to schválil a já jsem informoval vývojáře a testery o tomto chování tohoto nástroje.

5 Uplatněné a chybějící znalosti

5.1 Uplatněné znalosti

Během odborné praxe jsem uplatnil mnoho znalostí, které jsem získal na vysoké škole. Samotné programování jsem se naučil v předmětu Programování I, objektové programování v předmětu Programování II a samozřejmě v předmětu Programovací Jazyky, ve kterém jsem získal základy objektového programování v Javě a mohl se tak orientovat ve zdrojových kódech projektu. Využil jsem znalosti SQL z předmětů Úvod do databázových systémů a Databázové a informační systémy, díky kterým jsem si mohl napsat složitější dotazy na databázi nebo zkontrolovat správnost vrácených dat z kódu v Oracle SQL Developer. Jako poslední jsem využil naučený javascript z předmětu Tvorba Aplikací pro Mobilní Zařízení I.

5.2 Chybějící znalosti

Největším nedostatkem z mého pohledu byla praxe s takto velkým projektem, ale díky ochotným kolegům z týmu si myslím, že jsem tuto mezeru ve znalostech vytěsnil a doplnil. Dalším nedostatkem byla neznalost Javovských frameworků, jako Hibernate a Spring, které se v projektu využívaly. Také jsem nikdy nepracoval s verzovacím nástrojem SVN, ani buildovacím nástrojem Maven. I když jsem měl základní znalost javascriptu, tak jsem se musel naučit základy AngularuJS. Práce s ním je rozdílná proti práci s čistým javascriptem nebo knihovnou jQuery.

6 Závěr

Tato práce měla za cíl popsat odbornou praxi, kterou jsem absolvoval ve firmě Tieto Czech na pozici Junior Java vývojáře. Získal jsem zde cenné praktické zkušenosti z vývoje velkého projektu v korporátním prostředí a jak funguje nejenom vývoj takové aplikace, ale také firma samotná. Bylo mi umožněno rozšiřovat aplikaci o funkcionality a opravovat chyby v aplikaci, což mě posunulo v mých programovacích schopnostech. Také jsem se dozvěděl, že komunikace v týmu tvoří velkou část této profese i mé praxe. Zjistil jsem také, že vývoj v týmu má kromě výhod také nevýhody. Například různorodost kódů, pohledů na problémy a jejich řešení. Ale i přes tyto problémy, práce v týmu ulehčuje a urychluje vývoj aplikace.

Na závěr musím svoji odbornou praxi zhodnotit velmi pozitivně, protože mi dala znalosti, které jinak během studia nelze získat a do budoucna plánuji dále se podílet na vývoji tohoto zajímavého projektu s mým týmem.

Literatura

[1] Tieto History [online] [cit. 2018-04-15]. Dostupné z:
https://www.tieto.com/about-us/history-tieto

[2] Tieto Czech Historie [online] [cit. 2018-04-15]. Dostupné z:
https://www.tieto.cz/tieto-o-nas/historie-tieto-czech-republic

[3] Cross-advertising [online] [cit. 2018-04-15]. Dostupné z:
https://www.tieto.com/industries/media/cross-advertising-it-solution-media-houses

[4] Oracle Java history [online] [cit. 2018-04-15]. Dostupné z:
http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html

[5] About Java [online] [cit. 2018-04-15]. Dostupné z:
https://go.java/index.html

[6] AngularJS [online] [cit. 2018-04-15]. Dostupné z:
https://angularjs.org/

[7] TypeScript [online] [cit. 2018-04-15]. Dostupné z:
https://www.typescriptlang.org/

[8] FOWLER, Martin Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002